THE UTILIZATION OF PROCEDURE MODELS IN DIGITAL IMAGE SYNTHESIS

MARTIN EDWARD NEWELL

THE UTILIZATION OF PROCEDURE MODELS IN DIGITAL IMAGE SYNTHESIS

by

Martin Edward Newell

A dissertation submitted to the faculty of the University of Utah in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

University of Utah

Summer 1975

UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Martin Edward Newell

I have read this dissertation and have found it to be of satisfactory quality for a doctoral degree.

26 May 1975
Date

[redacted]
David C. Evans
Chairman, Supervisory Committee

I have read this dissertation and have found it to be of satisfactory quality for a doctoral degree.

MAY 7 1975
Date

[redacted]
Steven A. Coons
Member, Supervisory Committee

I have read this dissertation and have found it to be of satisfactory quality for a doctoral degree.

30 May 1975
Date

[redacted]
Richard F. Riesenfeld
Member, Supervisory Committee

I have read this dissertation and have found it to be of satisfactory quality for a doctoral degree.

13 June 1975
Date

[redacted]
Thomas G. Stockham, Jr
Member, Supervisory Committee

UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of Martin Edward Newell in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to the Graduate School.

23 JUNE 1975
Date

[redacted]
David C. Evans
Member, Supervisory Committee

Approved for the Major Department

[redacted]
Anthony C. Hearn
Chairman/Dean

Approved for the Graduate Council

[redacted]
Sterling M. McMurrin
Dean of the Graduate School

ACKNOWLEDGEMENTS

I wish to express my deep appreciation to David Evans as a source of continuous help and inspiration during the past two years.

I am also indebted to Ivan Sutherland for considerable help in formulating the background and motivation for this work, and without whom I would not have come to Utah in the first place; Steven Coons for many stimulating conversations; Rich Riesenfeld for help in preparation of the manuscript and for putting some of the key ideas on a firmer basis; Tom Stockham who has helped me to appreciate the wider relevance of this work; and Mike Milochik for photographic assistance over and above the call of duty.

In addition I am grateful to all the students and colleages whose interest and assistance have been an enormous help.

Last, but by no means least, I thank my wife, Sandy, for her constant support and understanding.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ABSTRACT

Many algorithms have been developed for synthesizing shaded images of three dimensional objects modeled by computer. In spite of widely differing approaches the current state of the art algorithms are surprisingly similar with respect to the richness of the scenes they can process.

One attribute these algorithms have in common is the use of a conventional passive data base to represent the objects being modeled. This paper postulates and explores the use of an alternative modeling technique which uses procedures to represent the objects being modeled. The properties and structure of such "procedure models" are investigated and an algorithm based on them is presented.

CHAPTER I

INTRODUCTION

For over a decade considerable effort has been expended in developing techniques for synthesizing visual images of scenes modeled by computer. This effort has advanced the state of the art from the generation of simple line drawings of two dimensional objects to the production of systems capable of synthesizing full color, real time, perspective, shaded, visible surface images of three dimensional objects with a startling degree of realism.

This paper is concerned with the synthesis of shaded pictures of three dimensional objects. Several algorithms have been developed for this task, the techniques used being many and varied. However, certain similarities can be drawn between the various algorithms. The scene to be rendered is modeled in the computer. The model is then processed by the algorithm, sometimes into another intermediate model, and is ultimately transformed into a picture. The types of models used vary, but they may all be described as "data base" models characterized by coordinate and structure information.

In the interest of limiting the scope of the algorithms, the data base is always constructed from a single primitive form, e.g. planar polygons, quadric surfaces, bivariate surface patches etc., planar

polygons being the most commonly used. In their paper which compares ten hidden surface algorithms all of which use a polygonal primitive representation, Sutherland et al (1) point out that all the algorithms studied have a similar capacity in terms of scene complexity and processing time.

In view of the widely differing techniques used, one is led to ask whether there is some common factor inherently limiting all these algorithms. This paper proposes that there is such a factor, and that it is in the form of the model used by all these algorithms, and indeed by the algorithms which use other passive primitive forms of representation. This proposal is based on the belief that the loss of information inherent in requiring that everything be represented in a common primitive form is a major factor limiting the capability of present approaches.

This paper investigates the use of an alternative technique for modeling, namely the use of active procedures for the representation of objects for the purposes of synthesizing shaded pictures. Models of this form will be called Procedure Models. Through the use of procedure models images of scenes one hundred times more complex than the previous practical limit have been generated.

Structure of the Paper

This paper may be divided into three main parts. Chapters II and III describe existing techniques for image synthesis, and are not

essential reading for an understanding of the other chapters by persons familiar with the field. Chapters IV through VII deal with procedure models and their relevance to three dimensional analyses. This part is of a general descriptive nature, and presents ideas of general applicability. Chapters VIII and IX describe a visible surface algorithm based on procedure models, and may be ignored if only the general ideas are of interest. Chapter X attempts a categorization of procedure models as used in digital image synthesis.

CHAPTER II

EXISTING VISIBLE SURFACE ALGORITHMS

There are many ways to categorize the existing published visible surface algorithms. This chapter does not attempt a complete categorization, or even a complete list of published algorithms, but is intended to provide some introduction and background to the techniques and terminology referred to in later chapters.

The process of generating a visible surface image of a scene can be divided into five tasks:

a) transformation of individual objects into correct positions in the scene

b) application of a perspective transformation to simplify many of the subsequent visible surface computations

c) clipping to remove parts of the scene whose images would lie outside the bounds of the display device

d) determination of the visible surfaces

e) rendering an image of the visible surfaces.

By using a homogeneous coordinate representation tasks a) and b) can both be implemented as matrix multiplications. Tasks a), b) and c) can be regarded collectively as scene preparation, and are discussed further below.

Task d), the determination of visible surfaces, is what largely

distinguishes between the various visible surface algorithms.

Task e) actually generates the image, a process which proves to be far more difficult than at first might be expected.

In some algorithms tasks d) and e) occur concurrently. They are separated here to assist in explanation. Likewise, task c) is sometimes not explicitly done, its effect being a part of tasks d) or e).

Scene Preparation

Excellent descriptions of the process of scene preparation for algorithms which operate on polygons are given by Sutherland et al (1) and so only a brief review is given here. Many of the techniques apply equally to other algorithms.

Transformations of objects, for the purposes of defining position, orientation, scale, and perspective, can all be implemented as matrix operations in homogeneous coordinates. The transformation of a point (x y z), in "object" coordinates can be effected by forming the product of the extended homogeneous vector (x y z 1) with the 4x4 compound transformation matrix:

$$(x\ y\ z\ 1)\begin{bmatrix} r & r & r & p \\ r & r & r & p \\ r & r & r & p \\ t & t & t & 1 \end{bmatrix} = (x'\ y'\ z'\ w') \qquad (1)$$

where the partition indicated with 'r' may be interpreted as the rotation and scaling, 't' the translation, and 'p' the perspective.

The resulting 4-vector must be divided through by w' to reduce it to three-dimensional "screen coordinates", (X Y Z). That is:

$$X = x'/w' \qquad Y = y'/w' \qquad Z = z'/w'$$

It is remarkable that the perspective transformation and subsequent division, an overall non-linear transformation, has the properties of preserving straight lines, preserving flat planes and preserving depth ordering. These properties permit the determination of visible surfaces to be carried out on the transformed and clipped objects as if only an orthographic projection were involved. The objects will have been distorted so that their orthographic projections are the same as the perspective projections of the given objects. This distortion is illustrated for a cube in Figure 1. The effect is to actually make distant objects smaller.

The division by w' to generate screen coordinates is susceptible to overflow, and in physical terms projects points which are both in front of and behind the eye. It is desirable not only to avoid overflow but also to remove those objects, or parts of objects, whose images would lie outside the limits of the display device. These problems can be avoided by clipping the transformed objects while represented in homogeneous form so that after division only the required parts will remain. For example, suppose the limits of the display device were -1 to +1 in X and Y, and the limits of interest in Z are 0 to 1, a convenient normalized range. Then the only parts of the objects which are of interest are those which satisfy:

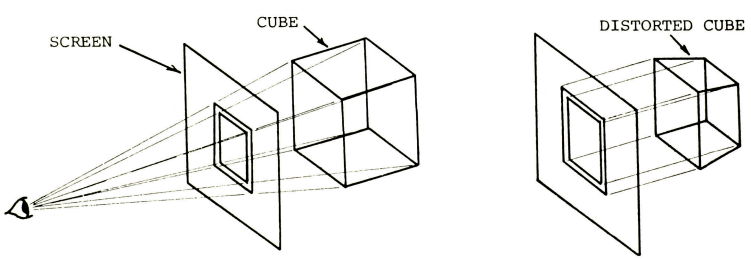
$$-1 < x'/w' < 1$$
$$-1 < y'/w' < 1 \qquad (3)$$
$$0 < z'/w' < 1$$

which for w' 0 gives:

$$-w' < x' < w'$$
$$-w' < y' < w' \qquad (4)$$
$$0 < z' < w'$$

These inequalities exclude all points for which $w'<0$, which is normally the required effect. A suitable algorithm for clipping polygons to the above limits is given by Sutherland and Hodgman (2).

For algorithms which represent objects as constrained quadric surfaces the clipping and perspective division need not be explicitly



a Perspective projection b Orthographic projection

Figure 1 Distortion of cube by perspective transformation

carried out. Quadric surfaces can be represented in the form

$$P.A.P^* = 0 \qquad (5)$$

and

$$P.Ci.P^* > 0 \qquad (6)$$

where A is the 4x4 matrix of coefficients of the surface, Ci are the matrices of coefficients of the constraining surfaces, P denotes a point, (x y z 1), on the surface, and P* its transpose. From equation (5):

$$P.(T.TI).A.(TI^*.T^*).P^* = 0 \qquad (7)$$

where T is any 4 x 4 non-singular matrix and TI its inverse. Regrouping:

$$(P.T).(TI.A.TI^*).(P.T)^* = 0 \qquad (8)$$

If T is interpreted as a 4 x 4 homogeneous transformation then

$$P' = P.T$$

represents points on the transformed surface. Therefore the matrix:

$$A' = TI.A.TI^*$$

must represent the matrix of coefficients of the transformed surface, which is simply another 4 x 4 matrix. This indicates that not only are lines and planes preserved by the perspective transformation, but also quadric surfaces.

Clipping of quadric surfaces can be effected by adding the relevant constraining planes written in the form of equation (6). Notice that this procedure avoids any explicit division to achieve screen coordinates, although generation of images of these surfaces implicitly involves division. Algorithms using quadric surfaces as their primitive form have been developed by Mahl (3) and MAGI (4).

Visible Surface Determination

An excellent review and characterization of ten hidden surface algorithms may be found in Sutherland et al (1). (The terms "hidden surface algorithm" and "visible surface algorithm" are often used interchangeably). The paper includes some hidden line algorithms, but restricts itself to algorithms which operate on objects represented by groups of planar, or nearly planar, polygons.

Sutherland et al divided the hidden surface algorithms considered into three classes: object space, image space, and list priority. The object space algorithms happen to be hidden line algorithms, which are not of primary interest in the present paper. The image space algorithms are further divided into area-sampling and point sampling, typified by the algorithms of Warnock (5) and Watkins (6) respectively. The list priority algorithms are subdivided into a priori and dynamic, examples being the algorithms of Schumaker et al (7) and Newell et al (8). A brief description of each of these algorithms will be given.

One of the earliest hidden surface algorithms was that of Warnock, which is classified as area sampling, image space. This algorithm may be described in terms of the technique of breaking down a large problem into several smaller problems whose solutions may be readily determined. In Warnock's algorithm the large problem is that of generating an image of the entire scene, or at least that part of it which lies on the screen. The smaller problems whose solutions may be readily determined are the generation of images of simple scenes.

The breaking down of the large problem into the smaller ones is achieved by subdividing the scene with planes through the eye, so that the resulting sub-scenes will not overlap and can therefore be treated independently. Variations on the algorithm may be achieved by using various definitions of a simple scene, and by using various subdividing schemes. One well known combination of these variables defines a simple scene as either being one containing no polygons or being a single polygon which fills the viewing area. The basic subdividing scheme simply quarters the screen into four equal parts. Subdivision is terminated when either a simple scene is achieved or the viewing area is the size of a single resolvable picture element. The two main problems with Warnock's algorithm are the comparatively expensive subdivision of the scene, and the fact that output is not generated in a convenient order for display on a raster scan device.

The point sampling image space algorithms are typified by that of Watkins, which is a development of two earlier algorithms, those of Romney et al (9) and Bouknight (10). All these are scan-line algorithms, a term which refers to the fact that they all generate the image one scan-line at a time. This is extremely convenient for display using a raster scan device. The generation of the image on each scan-line is achieved by considering the intersection of the scene with a horizontal plane through the eye and containing the scan-line. This reduces the problem to a two dimensional "hidden line" problem on the plane, the "lines" being the intersections of polygons with the plane. It is convenient to solve this problem in a left to right fashion to generate the individual picture elements in

an order suitable for display. The techniques used to solve this problem constitute the major differences between the three scan-line algorithms. Bouknight and Watkins made use of the observation that the ordering of edge crossing usually changes very little from one scan-line to the next and so the solution on one scan-line can be computed incrementally from the solution on the previous scan-line. Romney observed that this should be possible but failed to capitalize on it. This technique is referred to as scan-line coherence and permits significant savings of computation time. Watkins algorithm, and the associated scene preparation has been implemented in special purpose hardware which can generate images of 2000 edge scenes at 30 images per second.

The list priority algorithms, which are partially image space and partially object space, operate by establishing a priority list of polygons. One polygon has a higher priority than another if it obscures the other. In a loose sense high priority polygons are near the eye. The priority algorithms of Schumaker et al and Newell et al differ in the ways in which the priority ordering is computed, and in the way it is used.

By putting certain restrictions on the scenes which could be processed, Schumaker was able to generate the priority list for a sequence of views very simply, although a considerable amount of view independent work had to be done before any images were generated. This fitted in with Schumaker's aim to develop a simulation system for producing a real time sequence of views of a largely unchanging scene.

Priority determination uses two ideas which Schumaker refers to as clusters and linear separability. Clusters are groups of polygons which, after the removal of back facing, and therefore invisible, polygons (assuming solid objects), have a priority ordering independent of the view point. A simple example of this remarkable phenomenon is any closed convex polyhedron, since the front facing polygons cannot overlap and so the priority order is arbitrary. Linear separation involves dividing the scene into convex cells with a collection of planes such that each cell contains only one cluster. Inter-cluster priority is then determined by finding which cell contains the eye, a process which grows linearly with the number of dividing planes. To generate an image, each polygon is represented by its edges. For each edge it is determined whether the scanning spot on the television display is on the inside or outside of the edge. When the spot is found to be on the inside of all edges of a given polygon, that polygon is considered to be potentially visible at that spot. The polygon chosen for display at the spot is the potentially visible one having highest priority. This algorithm was the first one implemented in hardware to produce pictures at 30 frames per second. The number of polygons it can process is largely dependent on the number of edge processors that can be afforded since all the edge calculations must be carried out in parallel. As a software algorithm it is rather slow.

The priority algorithm of Newell et al imposes no special conditions on the scenes it can process. The priority list is newly constructed for each image, as follows. The polygons are first

ordered by the distance from their farthest point to the eye. The polygon having the greatest distance is probably of lowest priority. This ordered list is then checked and modified, by reordering and polygon splitting, to transform it into a true priority list. In order to generate an image Newell's algorithm uses a frame buffer, a device capable of digitally storing one frame of picture. By writing images of polygons into the frame buffer in reverse priority order the correct image of the scene is created. Removal of hidden surfaces is achieved by overwriting in the frame buffer by higher priority polygons. The main problems with this algorithm are the computational expense of establishing the correct priority list, and the need for a frame buffer.

Visible Surface Rendering

Once it has been established which surfaces, or surface fragments, are visible it is necessary to generate images of those surfaces. For the scan-line algorithms this rendering occurs one picture element at a time, simulataneously with the determination of the visible surfaces, whereas in Newell's algorithm it is required to render whole surfaces one at a time. This distinction can have an effect on the techniques used to render an image of a surface.

The simplest form of rendering, or "shading" as it is often called, involves assigning a fixed shade, or color, to each surface. All picture elements representing that surface are then given the same

shade. This approach, while simple, does not take into account the position of any light sources and can make objects appear to be illuminated internally since the shade of a surface is independent of its orientation.

An improvement over the fixed shading can be achieved using Lambert's cosine law of illumination. According to this law the density of illumination of a surface is proportional to the cosine of the angle between the normal to the surface and the direction of illumination. This simply states that the illumination of a surface is greater the more nearly it directly faces the light. If it is assumed that the perceived intensity is proportional to the illumination density, a phenomenon known as pure diffusion, then a more realistic shading rule is realized. Objects take on the appearance of paper or a similar matte surface. The illusion of a shiny surface can be achieved by using the cosine of the angle of incidence to some power. This has the effect of making the orientation required to give a surface maximum illumination much more critical, and hence gives the appearance of highlights.

For curved surfaces approximated by an array of planar polygons the above techniques do not yield acceptable results. This is because even though the approximating surface is continuous in value, being discontinuous in first derivative, the resulting intensity distribution is discontinuous in value, Figure 2, and is therefore a very poor approximation to the correct continuous distribution. This situation is aggravated by the fact that the eye accentuates

discontinuities in intensity, known as the Mach band effect. Gouraud (11) sought to remedy this problem by using a linear interpolation of intensity rather than the step function implied above. The aim was to achieve an intensity distribution which was continuous at least in value. Instead of using normals to the polygons to compute intensity Gouraud uses normals at the vertices of polygons. These normals are either known from the original curved surface or can be approximated. The vertex intensities are then interpolated using a simple linear interpolation illustrated in Figure 3.

Gouraud shading, while realizing startling improvements in the images of approximated curved surfaces, does suffer some problems. If the number of vertices in a polygon is greater than three then in

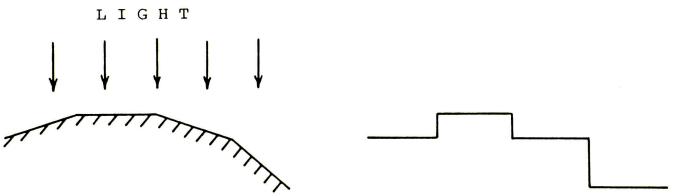
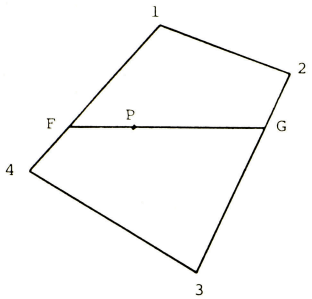


Figure 2 Cross section through approximated curved surface

general the resulting shading is axis dependent, though in practice this has not proved to be a major problem. A more serious problem, illustrated in Figure 4, can give rise to areas of constant shading on a curved surface. This occurs if the intensities at the vertices of a polygon are all the same, even though the normals are not parallel. The situation is aggravated if an attempt is made to simulate highlights. This is because the level of detail allowed in the intensity distribution is restricted to be no greater than the geometric level of detail.

The above problems have been largely eliminated by Bui-Tuong (12), who, instead of interpolating intensity, interpolates surface normals. Each of the three normal components is linearly interpolated using the same rule as that used by Gouraud. Having thus established a normal for each point on the surface, the intensity is calculated. Computationally this is a much more expensive process involving normalization of the interpolated normals and computation of intensity at every displayed point. However, the process does lend itself to special purpose hardware implementation. Bui Tuong also investigated more realistic methods for computing perceived intensity at a point. By considering the physics of the situation he was able to develop a realistic model incorporating the reflective and dispersive components of the perceived illumination.



$$I_p = a.I_f + (1-a).I_g$$

$$I_f = b.I_1 + (1-b).I_4$$

$$I_g = c.I_2 + (1-c).I_3$$

Figure 3 Gouraud interpolation of intensities

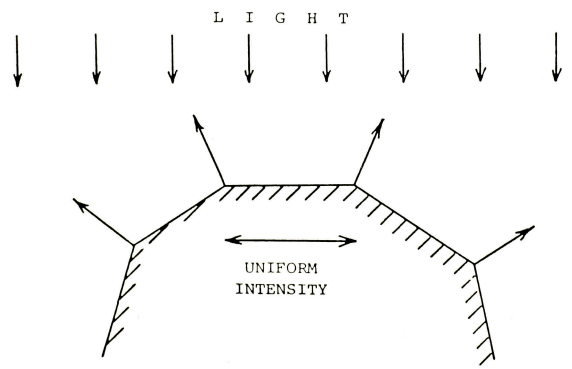


Figure 4 Constant intensity on approximated curved surface

CHAPTER III

LIMITATIONS OF CURRENT TECHNIQUES

Limitations on the complexity, or richness, of scenes that can be processed by existing techniques are due to several causes. These can be categorized three ways: storage requirements, processing requirements, and restrictions on the scene.

Storage Requirements

In the case of algorithms which represent objects as collections of polygons, a serious limitation is simply the volume of data which must be handled. For example, a scene consisting of 1000 quadrilateral polygons defined in terms of a set of 1000 points requires 7000 words of storage. The task of storing scenes 10 or 100 times this size in fast memory exceeds the capacity of many computers in use today.

However, it is not always meaningful to separate storage and processing requirements, in that given sufficient magnetic tapes, for example, any amount of information can be processed, albeit in an unacceptable amount of time. It is perhaps more meaningful to consider the pattern of accesses to the data, and to determine how

much of the data needs to be held in fast storage if the use of fast storage is to have a significant impact on the processing time.

In this regard, Warnock's algorithm has problems in that the parallel nature of the algorithm requires all the data to be held in fast memory. Watkin's algorithm is much better in that only the list of currently active edges need be held in fast memory for substantial gains in speed to be realized. Accesses to the bulky y-sorted edge list are serial and occur once per scan line. The algorithm of Newell et al, like that of Warnock, requires access to all the data in a random order, including accesses to the frame buffer. Therefore the addition of a small amount of fast memory gives no great advantage.

It seems that the storage requirement problem can be handled in two ways. One way is to find some means for serializing accesses to the data so that the bulk of the computations can be performed in the restricted amount of fast memory available. Recent work by Sutherland has yielded significant advances in this direction. Another way is to develop techniques for representing the necessary information in a more compact form.

Processing Requirements

The second major factor limiting the capabilities of current techniques is computation time. For nearly all known visible surface and visible line algorithms the time taken to generate an image grows faster than linear with the complexity of the scene. This fact makes

some algorithms unreasonable for even relatively modest scenes. Roberts' (13) algorithm, for example, becomes prohibitive for scenes containing more than a few hundred polygons. The current state of the art algorithms have succeeded in reducing the impact of the nonlinear effects enough to allow the handling of scenes of sufficient complexity to be of use in other than academic applications. However, for those algorithms which represent scenes as collections of polygons the current practical upper limit on complexity is in the region of 2500 polygons.

For those algorithms which use forms of representation other than polygons, for example quadric surfaces, the equivalent useful complexity seems to be no better, although individual objects may appear more pleasing. These other algorithms benefit from requiring fewer primitive forms to describe any given scene, but the escalated difficulty of dealing with each primitive often outweighs the potential gain.

Restrictions on the Scene

The third major limitation found with existing techniques is concerned with restrictions on the scene. This has several aspects, the first of which involves constraints on the scene resulting from assumptions or simplifications in the algorithm. Examples of such constraints include non-intersecting objects, convex polygons, and a limit on the number of edges allowed per polygon. The constraints

imposed by Schumaker's algorithm are unusually severe, although they are acceptable in simulation applications involving fairly static scenes.

The second aspect involving restrictions on the scene stems from the observation that all existing algorithms require that the scene be represented as a collection of instances of the same primitive form, e.g. polygons, quadric surfaces, bivariate patches, etc. This brings about a certain simplification of implementation and allows each algorithm to exploit the convenient properties of its chosen primitive. However, no one form is optimal for representing all scenes. Polygons can be used to approximate virtually any shape, but questions such as how many should be used to represent any given curved surface have no satisfactory general answer. This is because the minimum number of polygons needed to give an acceptable approximation to a curved surface is dependent, among other factors, on the view of that surface. Near objects need to be approximated more accurately than distant ones. Similar comments apply to the inclusion of fine detail which may only require a crude representation when in the distance. In practice sufficient polygons are used to give adequate representation for the worst case expected, which implies a wastefully detailed representation of objects in the distance.

Quadric surfaces are ideal for objects having conic generators, and can be used in a piecewise manner to approximate more general surfaces. However this task is quite difficult, and can generate many

fragments. Experience indicates that scenes modeled with quadric surfaces tend to look somewhat stylized, being made from spheres, cones, cylinders, etc.

Bivariate patches are extremely versatile and can be used to represent a wide class of curved surfaces. As with quadric surfaces, techniques for representing any given shape are not straightforward, although recent research in this area by Riesenfeld (14) and others has made significant advances. This has led to formulations of piecewise polynomial and rational polynomial patches which have been designed specifically to facilitate the representation of arbitrary curved surfaces. It would seem desirable to have a visible surface algorithm which could make direct use of these new formulations. Catmull's (15) recent work provides an example of one possibility in this direction. Perhaps more to the point is the fact that these forms are only suited to representing smooth surfaces and their use in representing planar faced objects can be extremely inefficient.

A further disadvantage of representing all objects by a common form is that much useful information regarding the coherence of objects is often lost. Indeed, most algorithms which use a polygonal representation treat each polygon as a separate, independent entity. All information regarding grouping or connectivity is either destroyed or ignored. This can be likened to doing a jigsaw puzzle with all the pieces kept face down. Failure to use such information is not a necessary consequence of using polygons, but the difficulties involved in making beneficial use of this information are severe enough to have

discouraged most investigators. A noteworthy exception is the use made by several algorithms of silhouette edges as distinguished from interior edges.

The third aspect of the question of restrictions on the scene is concerned with generality of representation. Many systems dictate a data format based on their particular primitive form. As these systems are developed, the need for more and more generality in the facilities provided results in escalations in the complexity of the data format until it begins to resemble a programming language. At this point the arguments for and against special purpose programming languages become relevant. The principal argument for such languages seems to be that specialization allows a user to specify what is wanted more directly and concisely. However, except in extremely specialized applications the special facilities tend to be overshadowed by facilities found in most general purpose programming languages.

An alternative approach is to provide a general purpose language, with specialized primitives imbedded either in the form of extensions to the language or by subroutine calls. If the language chosen is compatible with that used to implement the visible surface algorithm then the data description routines can be loaded together with the visible surface program to form a special purpose program for generating images. Several systems providing this facility have been implemented, but they still suffer from the requirement that the interface to the visible surface routines be strongly oriented towards

the primitive form used by the chosen visible surface algorithm.

These considerations point to the need for a system structure which allows the combined use of a variety of primitive forms in a way which is sufficiently flexible to allow the peculiarities of each form to be fully exploited. It should not impose unnecessary constraints on the range of facilities provided, and should provide for all the primitive forms currently found to be useful.

CHAPTER IV

PROCEDURE MODELS

Most data processing systems can be described in terms of a set of data, which represents the items to be processed, and a program which has encoded into it all of the processes to be applied to the set of data. In the particular case of digital image synthesis the term 'set of data' might be replaced by 'scene description' and 'program' by 'visible surface algorithm'. If one has several sets of data to be similarly processed then one need only generate the program once and apply it separately to the several sets of data. Such an organization is conceptually simple, the idea of representing an item by a collection of numbers being readily acceptable. For example, the representation of an object by a set of points each one represented by its x,y, and z coordinates, and a set of polygons each one represented by a list of points, is a widely used structure.

Passive Data Bases

Cases arise where the simple division of a system into active processor and passive data is inadequate. Typical shortcomings of such an organization arise from the need for parameterized instances of a prototype, the need for specifying a repetition of some data rather than actually repeating the data, the need for performing some

arithmetic to efficiently specify an item, and the need to specify conditional circumstances where the determining factors are external to the data. These shortcomings can be rectified by escalating the facilities provided by the data format which describes the items to be processed. When this is done the input data format is transformed from a list of numbers to a command language, or even to a resemblance of a general purpose programming language with subroutines, repeat loops, expressions, conditional operators, etc. The input data may then be viewed as a program which will be executed interpretively by the data input routines, the result of that execution being data to be processed by the main body of the processing algorithm.

There is, however, a more fundamental shortcoming to this segregation of processor and data. This arises when it becomes desirable to use widely differing processing techniques depending on what the data represents. This implies that the processor must cater for all possible types of data. Even in cases where the range of types is known in advance this can generate an unwieldy organization.

As a simple example consider a system for finding the geometric extrema of objects. Suppose the objects of interest are: groups of polygons, spheres, and bicubic patches. The input data format might have a herald for each object announcing its type, followed by a list of parameters. The 'interpreted language' approach might 'execute' such an input and produce a list of objects each one represented in a common format, for example, polygons. The processor would then operate by searching all the points on each converted object in order

to find its extrema. This approach gives a simple main processor which has to handle only one type of representation.

Definition of a Procedure Model

A more efficient approach to the previous problem using the idea of type-dependent processing might employ three algorithms, one for each type of object. The first, for polygonal representations, would search as above. The second, for spheres, would take the center of each sphere and simply add and subtract the radius to find the extrema. The third, for bicubic patches, might operate by repeated parametric subdivision of the patch to find the extrema within some given tolerance.

It might be argued that the only difference between this approach and the previous one is that the type-dependent processing has been moved from the data input routines into the main process. The difference, however, is rather more profound than this, in that each object may now be considered to be modeled by a procedure with which another procedure may interact. As an example suppose the goal in the previous example had been to find the volume of the minimum rectangular box containing each object. The main process would compute the product of the differences of the extrema of each object in each of the three coordinate directions. The way in which the main process finds these extrema may now be viewed as a question to each of the models themselves, rather than as an analysis of each object. The

method used by each model to answer that question is of no interest to the main process, so each model may use whatever technique it prefers to answer the question. Such representations of objects are examples of procedure models.

More formally, a procedure model is a model which represents its subject as a procedure with which other procedures can interact. The procedure model may be with or without parameters. Interactions with such a model are in the form of messages and include commands (e.g. 'output yourself'), and questions (e.g. 'what are your extrema?'). Responses can be confirmation of completion of some requested action, return of requested data, or an indication of failure to do one of these.

Properties of Procedure Models

The advantages of using procedure models stem from the higher level of representation they afford. A simple interpretation of this allows access functions to be built in with the conventional structure. Access to such a model is then carried out at a higher level than the manipulation of addresses, pointers etc. The access functions need not, indeed should not, know anything of the technique used by the model to derive requested information. If it becomes necessary to replace one form of the model with another then, provided that the interface to the model is sufficiently independent of the representation used, such a change can be made without requiring any

change outside the model itself.

Designers of data bases sometimes refer to the distinction between what is internal to the model and the external interface by refering to the physical data structure and the logical data structure. The physical data structure is the actual structure used to store information and is concerned with the words, pages, disk accesses etc. actually used. The logical data structure is some pseudo structure simulated by the model, and manipulated by the access functions. The mapping of logical data structure onto physical data structure is the function of the access functions which, together with the data, make up the model.

Although procedure models can be described in similar terms, such a data-oriented view obscures some of their most important attributes. One of these attributes is the freedom to partially, or totally, replace data with procedure. The case given in the previous chapter of finding the extrema of an object exemplifies the use of this freedom. If the model represents a sphere then given the position and radius as parameters the extrema can be generated by a simple arithmetic computation. The applicability of this technique is more widespread than may at first seem apparent. Several examples are given in Chapter IX. Even in cases where data is empirical and obeys no known law, the replacement of data by procedure can give significant savings. This can range from the provision of an interpolation rule for supplying intermediate values, to the fitting of a parameterized mathematical formula to the data and then storing

only the fitted parameters and a procedure for evaluating the formula. This is a well known and widely used technique, but the grouping of parameters with evaluation procedure into an entity whose structure cannot be seen from outside is not so widespread, and is the key to generality and modularity.

Another important attribute of procedure models is generality of parameterization. In addition to variables such as size, color, orientation etc. procedure models allow parameters which may have a drastic effect on the form of the item being represented. For example, a highway design system might use a procedure model to represent bridges. An important parameter to such a model would be the length of the bridge, not only to determine size but also as a type parameter to determine whether a suspension, beam, cantilever or arch bridge is required. A question to such a model might request a cost estimate, in which case the interrogating process may not be interested in what type of bridge is involved.

Since procedure models are executed as procedures they may embody any known data representation scheme. This means that conventional data structures form a subset of the class of representations allowed by procedure models. An important consequence of the higher level of representation is the ability to use several different models in one program. This allows each model to exploit whatever properties it chooses in order to carry out its function most efficiently, whether in terms of space, speed, or generality. However, if more than one type is used then it becomes necessary to define the interface to each

model in a model-independent way. This is very similar to the requirement in some systems that all data be represented the same way but here the requirement is, or can be, at a more abstract level. The interface need be concerned only with the information required by the access routines, not with the form of the models involved.

The choice of this interface can have a marked effect on the success of the resulting system. If the interface is chosen to be too low level then a strongly procedural model may need to generate unnecessarily detailed information. On the other hand, if the level is too high the possibility arises of requiring the model to do more than is necessary. As an extreme example of this an entire system could be considered to be a procedure model representing all the items to be processed, and responding to the one command: 'solve the problem'. The choice of the right interface depends on the types of items being processed and on the type of processing to be done.

The argument has been made that there is really no difference between passive data structures and procedure models, in that they are both stored as strings of bits in the memory of a computer. Furthermore, they are both interpreted, albeit by hardware in the case of procedure models, and so in practice they are effectively the same. Arguments such as these miss the concept of procedure models in that although they can indeed be viewed as a difference in degree, that difference is so great as to be 'transactionally different', a phrase which refers to a sufficiently great change of degree to imply a difference of type.

Relation to Other Work

It is relevant to see how the idea of procedure models relates to the more general description of representing entities by computer as studied in computer science. The implementation of structured programs involves building a hierarchy of virtual machines, each one using the primitives presented by lower levels, right down to the basic hardware machine. The data manipulated by such programs may be viewed at various levels, the higher level interpretations being derived from lower level ones via the relevant access functions. In some sense it is an arbitrary decision as to where the line is drawn between what is considered to be the process and what is the model being processed.

One of the principal virtues of structuring a program in this way is that the implementation of the access functions defining any given level can be changed without requiring any change to the higher level processes. Equivalently, an access function providing an interface to multiple lower level data types can be implemented, thereby allowing the use of several independent representations of the data type chosen by the implemented process. This consideration implies a preference for making the conceptual interface between process and model at as high a level as possible to permit maximum flexibility for modification of the model representation. Conversely, the higher the level of model representation then the more specialized the implemented process tends to become.

It would seem that a reasonable criterion for the choice of level of access function to the model is that it should reflect the level of item considered to be the basic data type accessed by the process.

Relating this criterion back to the interests of digital image synthesis, it is proposed that the process be described in terms of synthesizing an image of a scene which consists of a collection of individual objects. Therefore, the interface between general purpose process and model should be in terms of objects, rather than in terms of some lower level primitive. The resulting system will be specialized in the sense that it will only be capable of synthesizing images of collections of objects. But this is precisely the goal originally set, and thus is entirely appropriate.

The ideas and motivations leading to procedure models are not new. Examples of related work in other areas may be found in Hewitt et al (16), Winograd (17), Birtwistle et al (18), and Smith (19).

CHAPTER V

RELEVANCE TO IMAGE SYNTHESIS

The ideas behind procedure models are not new in digital computing. The notion that information may be procedurally generated when needed, as opposed to simply being retrieved, is widely used. The syntactic similarity between subscripted variable references and function invokations in many high level languages exemplifies the interchangibility of data and procedure.

Some of the notions of procedure models are not new to computer graphics. Newman's display procedures are an example. Their function is to replace a conventional numerical data base with a procedure for the purposes of generating line drawings. In some cases the ability to determine when an image would be entirely outside the viewing area is used to completely avoid execution of the procedure. However, display procedures do not normally interact with the calling process in order to generate an image, but tend to be passive image generators. This passive role is possible because display procedures are concerned with generating line drawings, an essentially serial process. In contrast, the present work is concerned with the generation of visible surface images of complex scenes, and in the use of procedures to represent the objects in a way which facilitates this non-serial process. In the area of digital image synthesis procedure models give several advantages over conventional data structure

models. These advantages are concerned with facilitating processing and with view dependence.

Object Coherence

Sutherland et al (1) note that all known visible surface algorithms capitalize on some form of coherence. The term coherence refers to the interrelation between certain processes or groups of operations. Such interrelations can allow considerable computational savings either by using the results of an analysis of one situation in another similar situation, or by replacing a group of operations by a single operation.

An example of the first type of coherence is the scan-line coherence used in Watkins' algorithm. The fact that, in general, the list of visible segments on one scan line is very similar to the list on an adjacent scan line is exploited. The list for any one scan line is computed as a pertubation of the list for the previous scan line.

An example of the second type of coherence is object coherence. If, for example, it can be determined that two objects are disjoint, then all parts of one object will be disjoint from all parts of the other, and no tests on individual parts need be performed. This observation can lead to significant savings. Of course, it will generally be more difficult to determine whether two objects are disjoint as compared with, for example, two polygons. However, if there are N polygons per object then there will be roughly N**2 times

as many polygon pairs as object pairs, and so even with a substantial escalation of difficulty overall savings can be made.

The use of object coherence can be extended to cover a range of analyses encountered in image synthesis. The potential gains arise from the handling of only a relatively small number of objects which can give savings in both computation time and space requirements. This implies that some compact representation for whole objects be used. The logical grouping of polygons into objects is not satisfactory since this does not alleviate the storage requirement problem.

The representation proposed here is the procedure model which allows objects to be modeled in whatever form is deemed most efficient. This does not, of course, exclude groups of polygons, and it allows alternative representations some of which may be highly procedural. The example already given of a sphere is a case in point.

Generality of Representation

The generality of representation afforded by procedure models is limited only by the level of interface chosen. Examples of representations relevant to image synthesis include: groups of polygons, potential surfaces, and various surface patch schemes including bicubic Coons patches and B-spline surfaces.

Higher level but more specialized representations are also

relevant such as groups of the above items and procedurally generated whole objects, such as ships or buildings. These higher level forms may resort to patches or polygons for the detailed representation of their subjects, but may be able to yield requested information or carry out certain operations at a much higher level.

Generality of Parameterization

In image synthesis typical parameters to models represented by passive data structures are such things as surface properties like color and reflectivity, and affine transformations to specify size, orientation and position. Procedure models permit a much more extensive parameterization of models simply by virtue of their procedural nature. Any variable that influences the represented object can be used as a parameter. Examples include non-affine transformations, angles, and key dimensions.

More general parameters can influence the actual form of the represented object. Office buildings can be characterized by the number of floors, the type of windows, and the type of roof. Level of detail can also be parameterized, although such a variable would probably be view dependent.

The generality of parameterization allowed by procedure models enables extensive use of instancing, since one model can represent a wide range of extensively differing objects. In the extreme this implies that one model could represent all objects in a scene using,

for example, polygons. This would be equivalent to the single representation schemes currently in use. In practice a compromise between this extreme and that of using a separate model for each object should be sought.

View Dependence

The representation of objects for the purposes of image synthesis is often done by approximation. For economy of computation, storage, and effort objects are frequently represented with only sufficient accuracy to appear acceptable in the anticipated views. This applies both to the level of fine detail represented and to the accuracy of approximation to curved surfaces by polygons.

In systems where objects are represented by passive data structures the level of approximation has to be chosen at the time the data structure is created. This leads to an inflexible relationship between object representation and viewing parameters which does give a degree of simplification and modularity. However, this approach requires that the level of approximation be chosen to be sufficient for the most critical view expected. The implication is that for most views objects will be defined in more detail than is necessary for an acceptable image.

Procedure models allow the representation of an object to be influenced by the view. Distant or small objects can be approximated more coarsely than near or large objects. Stated another way,

procedure models allow the degree of approximation for all objects to be equal in terms of image space coordinates.

Another aspect of view dependence relates to the silhouette edges of curved surfaces. Gouraud (11) and Bui-Tuong (12) have demonstrated techniques for generating images of curved surfaces from comparatively coarse polygonal approximations. These techniques can give an extremely good impression for the interior of curved surfaces but do nothing to improve the polygonal silhouette. View dependent procedure models allow a more accurate approximation to silhouette edges while retaining the economy of a coarser approximation for the remainder of the surface.

A further facility afforded by view dependence is the ability to remove an entire object from consideration if it is known to be invisible in the present view. This can occur in two ways - by the object being completely outside the viewing area, and by an object being entirely obscured from view in an easily determined way. The former has wide application whereas the latter is rather more specialized. An example of the latter is the case of a closed object either containing another object and having the eye outside, or containing the eye and having another object outside. In either case the other object is invisible.

CHAPTER VI

THE MODEL INTERFACE

It was stated in Chapter IV that the more abstract level of representation afforded by procedure models facilitates the use of several different types of model in one program. If types are to be mixed it becomes necessary to establish an interface to each model which is independent of internal model structure and to which all models can conform. The central procedures can then communicate with each model via this interface without concern for the internal details of each model.

Requirements

The factors which influence the choice of model interface are concerned with the type of processing to be carried out, and with the types of models anticipated. As was mentioned earlier, an interface which is too low level may result in much duplication of processing within the models, whereas too high a level may be unnecessarily restrictive or difficult to conform with.

For the purposes of digital image synthesis the model interface requirements are based on the considerations presented in Chapter V,

namely: object coherence, generality of representation, generality of parameterization, and view dependence.

The exploitation of object coherence requires that the relationships between whole objects can be analyzed without regard for the details of any one object. The technique proposed here to facilitate such an analysis is to replace each object with an enclosing convex polyhedron. The analysis will then be carried out on a set of convex polyhedra, and will exploit all the convenient properties of such simpler forms. For example, the task of discovering whether two objects are disjoint is implemented by searching for a plane having the two convex polyhedra wholly on opposite sides. Clearly, this coarse representation of objects may result in interfering convex polyhedra which represent non-interfering objects. This problem will be discussed in Chapter VIII.

For some analyses an even simpler representation of objects can be used, namely an enclosing convex polygon in two dimensional screen space. For the purpose of drawing hidden surface pictures the determination of whether the images of two objects of interest overlap is of interest. Given enclosing convex polygons to represent two objects, the determination of overlap is reduced to finding whether two convex polygons have any area in common. Again, the coarsness of such a representation can lead to apparent overlap where none exists.

The above considerations lead to the first requirement of the model interface. Each model should be capable of generating an enclosing convex polyhedron and an enclosing convex polygon in screen

space. Techniques for generating such polygons and polyhedra are given in Appendices A and B.

The second consideration relevant to digital image synthesis concerning the model interface is generality of representation. It is desirable that an object be represented by an appropriate form. This form is influenced by the need to be able to generate an image of each object in a compatible form if a picture of the whole scene is to be synthesized. Since the techniques required for generation of an image from any given model representation are model dependent, it is necessary for the model to be capable of generating an image of the object it represents. This is the second requirement of the model interface. The actual form of this image is dependent on details of the main processing algorithm and will be more closely defined in Chapter VIII.

The third consideration concerning the model interface is generality of parameterization. This simply implies that it must be possible to pass parameters to models which may be used in any way desired.

The fourth consideration dealt with in Chapter V was view dependence. This has largely been covered here by the requirement that each model should be capable of generating an image of its subject. This enables a variable degree of approximation to be used, and indeed, allows nothing to be generated if it can be established that the entire object is invisible.

The only other demands made on the model interface are that it must be capable of specifying the reading and writing of model parameters from and to secondary storage. This is necessary in order to define the scene being analyzed as well as to enable swapping onto secondary storage to be used if space restrictions dictate the need. The information actually transfered to and from secondary storage need not necessarily be in the form of program overlays. If use is being made of instances, then only the instance parameters need be transfered, the model procedure staying in main memory. This leads to the question of what structure is needed in which to embed the procedure models.

Structure

The operation of a simple data processing system might be described as follows. At initialization, the program is loaded into main memory and the data base is considered empty. During processing, external influences and internal computations cause data to be added to and taken from the data base. The notion of reading some data, which might represent an object, and adding it to the data base is quite straightforward.

However, if objects are represented by procedures, the direct analogy is that procedures should be read and added to a procedural data base. While such a system is quite feasible, and can be considered a use of program overlays, it is not a necessary

consequence of using procedure models. The high degree of parameterization allowed by procedure models implies that it is often not necessary to have a great number of different model procedures. For example, a city scene might use three types of procedure model - one for parameterized buildings, one for automobiles using surface patches, and one using polygon definitions for all the other items. Each object is then an instance of its procedural master and manifests itself as some data in a passive data base. This may sound remarkably like the conventional architecture mentioned earlier, but this is due only to an implementation detail. The idea that each object is procedurally described, in this case by a logical combination of instance parameters and corresponding procedure is still very much in evidence.

The use of instances can be carried a stage further. For example, given a procedure for dealing with objects defined using bicubic patches, a particular use of such a procedure could involve the parameters necessary to define an automobile. Another use could involve the parameters necessary to define a boat. If it were desired to generate a scene including several similar automobiles and several similar boats, then it would be desirable to use instances of the automobile and instances of the boat, the parameters of these instances being such things as position and color.

This implies two levels of instancing - the automobile and boat parameters can be considered instances of the bicubic patch model procedure, and the several automobiles and boats are instances of

these instances. This structure is illustrated in Figure 5.

It is relevant to ask where the idea that an object be represented by a procedure, called a procedure model, fits into this structure. In this case the procedure model for any one automobile is a logical combination of the bicubic patch procedure, the automobile parameters, and the instance of the automobile. Such a logical grouping is shown on Figure 5. For descriptive purposes these two levels of instances will be refered to as the model instance and object instance. Conceptually this still uses one procedure model per object, the apparent difference being due to the use of instancing as an efficient implementation device.

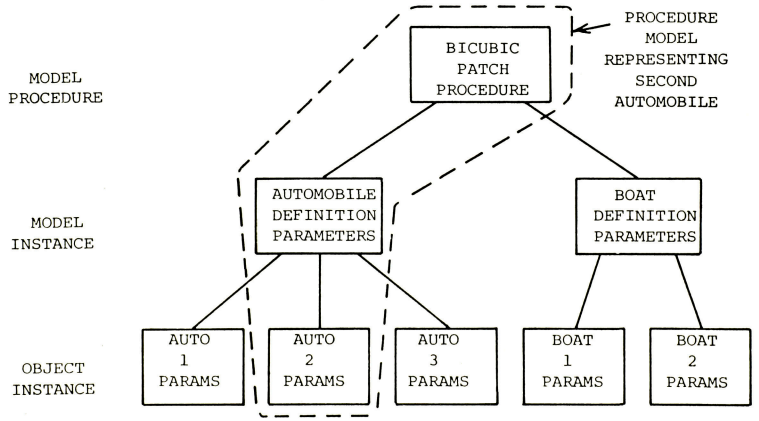


Figure 5 Procedure model structure

CHAPTER VII

ANALYSIS OF RELATIONSHIPS BETWEEN OBJECTS

The synthesis of images of modeled objects requires the determination of which objects, or parts of objects, are visible, and the generation of images of those visible parts. The former requirement necessitates the ability to determine which objects hide which others. This in turn implies the ability, given two objects, to determine whether one hides the other, either partially or wholly, and if so, which hides which. As was indicated in the previous chapter, the tests used in the analysis are carried out not on the objects themselves but on their enclosing convex polygons and polyhedra.

The tests are carried out in screen space in two groups: two dimensional and three dimensional. The two dimensional tests are carried out first in order to determine whether the two objects overlap on the screen, or rather, whether their enclosing convex polygons overlap. If it is determined that they do not overlap then there is no need to further analyze the relationship between them.

If the two dimensional tests indicate, to the accuracy afforded by the use of enclosing convex polygons, that the objects overlap it is necessary to resort to three dimensional tests. The goal of these tests is to find a plane which separates the two objects. If such a plane is found then the object which is on the same side of the plane as the eye is the obscuring object.

Two Dimensional Overlap

The two dimensional overlap tests are carried out on the enclosing convex polygons generated by the procedure models representing the objects. The first test used is called boxing. It involves conceptually constructing a minimum enclosing rectangle around each object polygon and then testing for overlap of the rectangles, see Figure 6. This takes the level of approximation to the original objects one stage further but has the advantage of being extremely simple, and if it indicates separation then the two objects are guaranteed to be separate. However, if it does not indicate separation then further tests are required to resolve the situation. The test actually involves four tests: if the minimum x of either box is greater than the maximum x of the other, and similarly in y, then the boxes are separate.

Of course, cases exist where the boxes overlap but the object polygons do not, see Figure 7. This necessitates a more thorough test to investigate separation. For this test appeal is made to the following theorem.

If two convex polygons in the plane have no area in common then at least one edge of at least one of the polygons is a segment of a line which separates the two polygons.

A proof of this theorem is given in Appendix C. An example is shown in Figure 7. To make use of this theorem it is necessary to

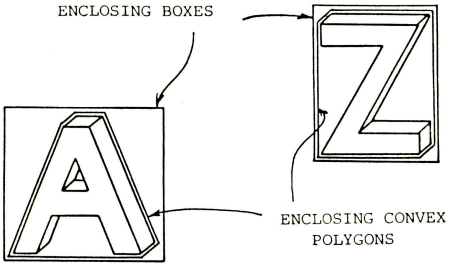


Figure 6 Enclosing convex polygons and boxes

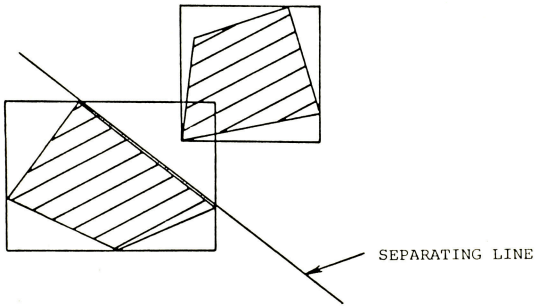


Figure 7 Non-overlapping polygons but overlapping boxes

consider each edge of each polygon and to test whether all vertices of the other polygon lie on the outside side of the implied line. If no edge is found satisfying the above condition then the two polygons overlap.

Simple Three Dimensional Separator Test

If the two dimensional tests indicate that the object polygons overlap then three dimensional tests are used to determine which object apparently overlaps the other. These tests involve searching for a plane having the two object polyhedra on opposite sides.

If the two dimensional polygons were derived directly from the polyhedra then there is no point in carrying out boxing tests in x and y. However a simple boxing test in z is worthwhile and operates in a manner similar to the x and y tests.

If boxing tests fail to establish separation then more stringent tests are needed. At first glance it might be thought that the three dimensional analog of the polygon separator theorem would state that: if two convex polyhedra have no volume in common then at least one face of at least one of the polyhedra is a region of a plane which separates the two polyhedra. However, although true in many commonly found cases this theorem does not always hold. A contradicting case is shown in Figure 8. Nevertheless, the fact that this theorem holds for many cases, and can be directly applied, is a motive to use it in an attempt to find a separating plane. The fact that it does not

always hold means that if its use fails to find a separating plane then it is not a consequence that no separating plane exists.

Use of this theorem is a direct extension of the two dimensional case. If, however, it is required to determine only whether object A obscures object B then it is only necessary to consider backward facing faces of A and forward facing faces of B. If no face is found satisfying the above conditions then further tests are needed to investigate the separation of the two objects.

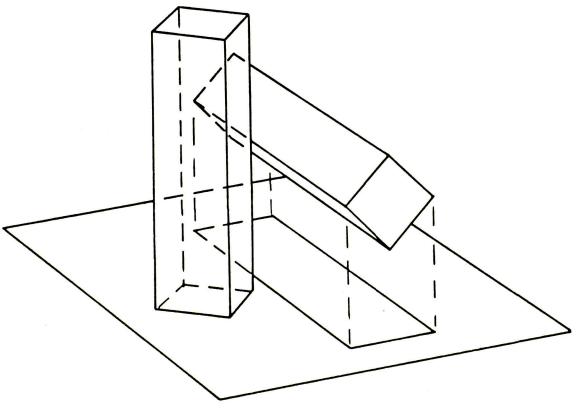


Figure 8 Separating plane not a polyhedron face

Comprehensive Three Dimensional Separator Test

The search for a plane which separates two convex polyhedra can be stated as follows. Determine a plane vector P = (a, b, c, d) such that its dot product with each vertex Ai = (Aix, Aiy, Aiz, 1) is positive, and its dot product with each vertex Bj = (Bjx, Bjy, Bjz, 1) is negative, where A is the set of vertices of one object and B is the set of vertices of the other. Symbolically:

$$P.Ai > 0 \qquad 1 \leq i \leq m$$

$$P.Bj < 0 \qquad 1 \leq j \leq n$$

These two inequalities can be combined by defining a new set of vectors C made up of A together with negated members of B. Then the requirement on P is that:

$$P.Ck > 0 \qquad 1 \leq k \leq m+n$$

This inequality may now be interpreted in a dual manner as follows. Find a point P which lies on the positive sides of all members of the set of planes C. The dimensionality is now one higher in that P is a 4-vector, and Ck can be considered to be 4 components of a 5 component plane equation in 4-space, the fifth component being zero. Hence, the search for P is equivalent to the search for a point in the solution region of a set of linear inequalities.

Various solutions to this problem have been proposed. The one used here is iterative and proceeds as follows. Set P to some initial vector. At each step of the iteration, for each vector, Ck, compute:

$$Lk := P.Ck$$

If Lk is positive proceed to the next member of C, otherwise adjust P

to a new vector:

$$P := P + f.Ck$$

where f is chosen to ensure convergence. The iteration stops when all Lk are found to be positive.

A reasonable initial setting of P is the plane which is half way along, and perpendicular to, the line joining the centroids of the given sets of vertices A and B, and having the centroid of A on its positive side.

The factor f is chosen as follows. The updating of P can be interpreted as moving the solution point P in a direction perpendicular to the plane Ck until it is moved onto the positive side of that plane. Indeed, the components of Ck may be considered to represent the normal to the plane in 4-space. To determine the required distance it is necessary to normalize Ck, such that Ck.Ck=1; then Lk represents the actual distance to the plane. In practice,

$$f = 1.05\ Lk$$

has been found to give good convergence, although any value of f greater than Lk will ultimately lead to convergence.

There is a problem with the iterative scheme described above. If a plane P which separates A and B exists then it will be found. However, if no such plane exists then the iteration will proceed indefinitely, and such a condition is difficult to detect. To overcome this difficulty a second iteration is run in parallel with that proposed. This second iteration seeks a point M which lies inside both convex polyhedra. The method used is essentially the same

as that described above, and is also carried out in a homogeneous space. In practice, each iteration takes steps in turn, and one of them is guaranteed to yield a decision, at which time both iterations stop.

There is an additional requirement on the search for the point M. This is that its homogeneous term must be positive. This is because a point M which satisfies all inequalities and having a negative homogeneous component is, in fact, outside all planes of both convex polyhedra, not inside. This may appear to be not possible but it must be remembered that the convex polyhedra may have been clipped and so they are not necessarily closed. A simple remedy is to add the plane vector (0, 0, 0, 1) to the set of polyhedron planes.

It has been found in practice that in cases where the simple three dimensional separator test failed and yet the polyhedra were separate, it can take several hundred iterations to find a separating plane. For this reason an upper limit (currently 100) is imposed on the number of iterations allowed. If no solution is found within this limit then it is assumed that the two convex polyhedra intersect.

CHAPTER VIII

A PRIORITY ALGORITHM USING PROCEDURE MODELS

The previous chapters have discussed procedure models, their relevance to digital image synthesis, and some techniques for their spatial analysis. This chapter describes a visible surface algorithm which uses procedure models, and which brings together all of the ideas and developments that have been described.

The algorithm to be described is a direct development of the priority algorithm of Newell, Newell and Sancha. The main developments concern the use of procedure models for the description of the scene and as the basic working elements, and the techniques used to determine priority. The advantages of this algorithm over existing ones stem directly from the use of procedure models, the main benefit being the ability to process scenes of complexity two decimal orders of magnitude greater than previously feasible.

Outline

The algorithm is a priority algorithm based in concept on that of Newell et al. A frame buffer is used to assemble the picture and to play an active role in the hidden surface elimination. The scene to

be portrayed is described as a collection of objects, each one represented by a procedure model.

The first phase of the algorithm reads descriptions of the objects making up the scene. The descriptions are in the form of instances of procedure models. Any modification to the positions or orientations of the objects are then carried out, and the viewing parameters are specified, e.g. eye position, field of view, etc.

The next function is the establishment of a priority ordering of whole objects. The generation of the enclosing convex polygons and polyhedra used in determining priority is not done all at once but takes place as each object comes into consideration.

The picture is assembled by writing images of the priority ordered objects into a digital frame buffer in reverse priority order, i.e. "farthest" object first. The generation of the correct image for each individual object is the responsibility of the model procedure generating that image. The removal of parts of the image of one object that are hidden by another is accomplished by the overwrite capability of the frame buffer, and is dependent on the establishment of a correct priority ordering.

When the entire picture has been assembled it is scanned out onto the display device. If the frame buffer is capable of distinguishing areas that have never been written then the background can be added at display time. Techniques of this type are more fully discussed in the section on frame buffers.

Establishment of Priority

A slightly abstract view of the task of generating a priority list of objects can be described as follows. Each object can be represented as a node in a directed graph, which will be called the priority graph. An arc exists between two nodes representing two objects if one object obscures the other, the direction of the arc being from the obscuring object node to the obscured. The problem of generating a priority list can be cast as the topological problem of mapping the nodes of the priority graph onto distinct points of a straight line such that all the arcs point in the positive direction along the line. Clearly, one necessary condition for this to be possible is that the priority graph contains no cycles. An example of a configuration of objects leading to a cycle in the priority graph is shown in Figure 9.

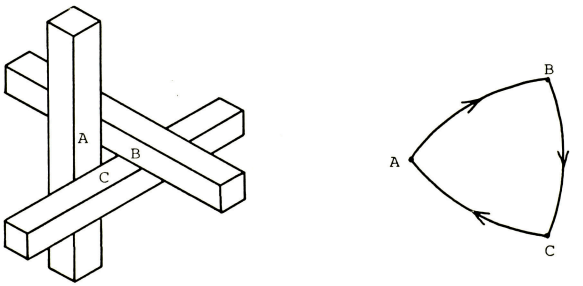
The priority graph, while useful as a representation for talking about priority, has not proved directly useful in creating an algorithm for generating priority lists. The technique used here is a development of that presented by Newell et al, the modifications being necessary to handle three dimensional objects rather than polygons. The technique attempts to minimize the number of detailed analyses of pairs of objects, and never actually generates the whole of the priority graph.

The method starts by asking each procedure model for its enclosing convex polyhedron, clipped to the viewing boundaries. This is used to find the extrema of each polyhedron in the z direction

(i.e. the viewing direction) and to set up various pointers used in space allocation. The polyhedron itself is then discarded. During this process a list of all objects not entirely off screen or otherwise invisible is generated. This list will be transformed into the priority list.

When the list of objects being considered is established it is sorted based on the furthest z value, Zfar, of each enclosing polyhedron. The direction of the list is such that the member having the nearest Zfar appears at the front of the list, and the one having the furthest Zfar appears at the end.

For scenes having well separated compact objects, the Zfar sorted



a Objects b Priority graph

Figure 9 Object yielding cycle in priority graph

list is often the priority list being sought. However, this is not always the case, as the example in Figure 10 illustrates. The sort would order the objects BA, whereas the correct priority ordering is AB.

The next phase of the algorithm checks out and, if necessary, modifies the list to transform it into a correct priority list. This can be done by working from either end of the list, but since the present purpose is to render objects into a frame buffer in reverse priority order, the processing is done by working from the end of the list corresponding to the farthest Zfar, thereby generating the priority list in order of increasing priority. At any stage the element at the end of the list is potentially the lowest priority element. This postulation is examined and if found to be true then the element is removed from the list and added to the priority list. The examination proceeds as follows.

The last element of the list P is compared with the set of elements Q whose Zfar is farther than the nearest z value of P. The set Q therefore contains all elements that could possibly be obscured by P. Figure 11 shows an example of P and the set Q.

In order to determine whether or not P obscures any member of Q the analyses described in Chapter VII are used until a decision is established. These tests are:

1. Two dimensional boxing
2. Two dimensional polygon overlap

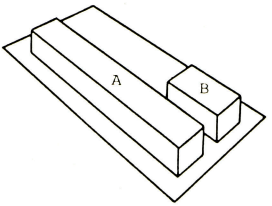


Figure 10 Zfar sorted list not correct priority list

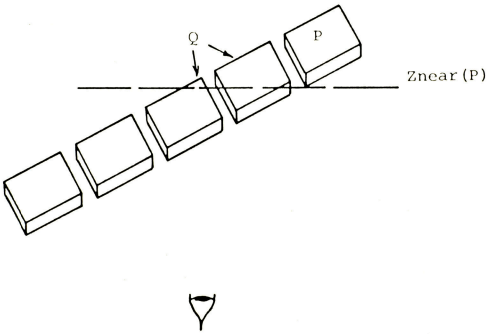


Figure 11 Element P and the set Q it overlaps

3. Simple three dimensional separator using back faces of Q and front faces of P
4. Comprehensive three dimensional separator

If these tests fail to establish that P cannot obscure a particular element of Q, Qi, then the possibility that P and Qi are in the wrong order is investigated. This only involves repeating test 3 using back faces of P and front faces of Q. Such a case was illustrated in Figure 10. In such a case Qi is moved to the end of the list and is treated as a new P.

Should it transpire that a reordered element belongs to a cycle in the priority graph, then the above procedure would, after a few more steps, attempt to reorder the same element again. To guard against this non-terminating possibility an element is marked when it is moved. If an attempt is made to reorder a marked element then it is concluded that a cycle exists, and a different course must be taken. This involves splitting the offending element into two or more pieces in an attempt to break the cycle. Such a procedure is discussed more fully in the section on Subdivision.

In the implemented version of this algorithm the fact that test 4 is much more costly than test 3 prompted investigation of reordering after test 3. If this failed then test 4 was entered, which actually tests for both possible orderings. To summarize, the complete list of tests carried out to determine whether P obscures any member of Q is:

1. Overlap in z (this defines a member of Q)
2. Overlap in x (2-D boxing)

3. Overlap in y (2-D boxing)
4. Overlap of polygons (2-D separator test)
5. P Behind back face of Qi (simple 3-D separator test)
6. Qi in front of front face of P (simple 3-D separator test)
7. Qi behind back face of P (re-order test)
8. P In front of front face of Qi (re-order test)
9. Comprehensive separator test

Clipping

Since the end goal of this algorithm is to synthesize an image of the scene it is necessary to remove from consideration all parts of the scene lying outside the viewing cone, see Figure 12. This is necessary for two types of reasons. The first is that the projection of points outside the viewcone can cause arithmetic overflow, and points behind the eye are erroneously projected. The second reason is not strictly necessary but is a matter of efficiency. It is clearly wasteful to analyze the relationship between two objects whose images will be entirely off the screen, or even to consider those parts of objects whose images will be off the screen. The question arises as to how, and at what stage, this clipping should be carried out.

The first obvious stage at which clipping could be done is before the enclosing convex polyhedra are generated, i.e. clipping the objects themselves. This is not desirable for the reason that it is not always convenient to split an object with an arbitrary plane. For

objects represented by groups of polygons the procedure is fairly straightforward, but for bicubic patches such a splitting is quite difficult and could yield several fragments none of which could be expressed in the same form as the original patches. One possibility is to transform the existing representation into a group of polygons and subsequently treat the object as a polygon object, for clipping as well as everything else.

The second stage at which clipping can be done is on the enclosing convex polyhedra. The polyhedra can be clipped in the coordinate system of the objects and then transformed into screen space. This is a well-conditioned operation, but has the disadvantage that the resulting polyhedra fragments may be larger than is

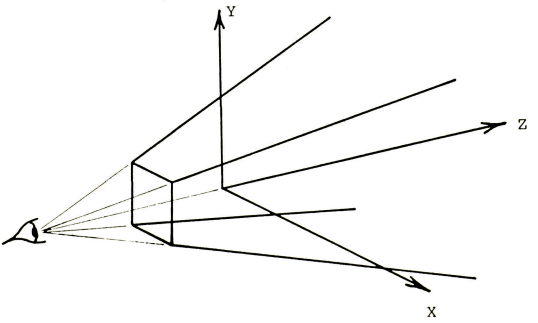


Figure 12 The viewing cone

necessary. Indeed, in an extreme case it is possible that although none of the object lies in the viewing cone, a part of the enclosing polyhedron might. In spite of this drawback the clipping of the enclosing convex polyhedra is considered preferable because it is independent of the representations used by the procedure models and therefore needs to be implemented only once.

The techniques for clipping a convex polyhedron will now be discussed. It is desirable to represent each polyhedron as a set of vertex vectors and a set of plane vectors, since both of these sets are used in the separation tests. The mathematics used here is very similar to that used by Sutherland and Hodgman (2). Indeed, a method for clipping a convex polyhedron is to consider the polyhedron as a set of polygons and to clip each polygon separately as described by Sutherland and Hodgman. The drawback with this technique is that it is difficult to avoid clipping every edge twice (since every edge is shared by two polygons) and to avoid storing every vertex as many times as it is used by a polygon, especially the generated vertices.

The key to avoiding this duplication of effort and storage is the edges of the polyhedron. This suggests that the convex polyhedron should be defined in terms of edges of the polyhedron, which in turn are defined in terms of vertices, Figure 13. This would facilitate keeping track of which edges have already been clipped though there is still a problem of whether interpolated or original vertices should be referenced by the edges. In view of these difficulties an alternative method for clipping polyhedra was developed, called the polyhedron

clipper.

The polyhedron clipper takes a more general view of the clipping process and expresses the end goal in terms of finding the intersection of two convex polyhedra, namely, the given polyhedron and the viewcone, which in screen space is a rectangular box, see Figure 14.

This symmetric view of the clipping process leads to certain simplifications but carries with it its own set of problems. The basic idea is that the polyhedron should be clipped by the viewbox, then the viewbox should be clipped by the polyhedron, and the results combined to form the intersection. The reason for clipping each volume against the other is that if the clipping of the polyhedron by the viewbox only actually clips its edges then newly formed vertices in the corners of the viewbox will be missed. Examples of these are indicated in Figure 14. This is precisely the problem addressed by Sutherland and Hodgman's polygon clipper. The solution proposed here, namely that of also clipping the edges of the viewbox by the planes of the polyhedron, is conceptually simple and yields the information required by the current algorithm. If it were necessary to build the actual polygons making up the clipped polyhedron then this method would require considerable extension. However, as was mentioned earlier, all that is required is a set of vertex vectors and a set of plane vectors. No new planes are generated and so plane vectors included in the clipped polyhedron are a subset of those in the given

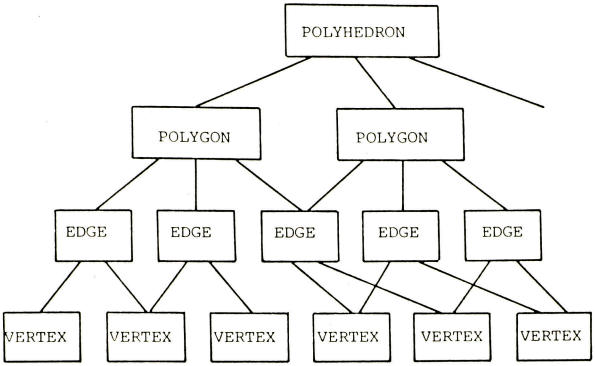


Figure 13 Polyhedron structure

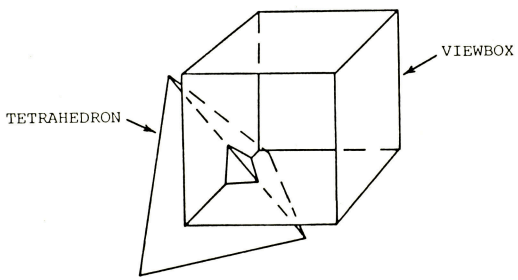


Figure 14 Intersection of tetrahedron with viewbox

polyhedron. A plane vector is included in the clipped polyhedron if either a part of an edge of its defining polygon is within the viewbox, or if it is the last plane to clip an edge of the viewbox. The latter condition is necessary in order to handle the case shown in Figure 15 where the clipped polyhedron contains no edge of the given polyhedron.

It may be illuminating to compare the proposed method for polyhedron clipping with the method mentioned earlier, namely, the use of the polygon clipper modified to avoid duplication of effort and vertices. The present method is not recursive, but will only handle convex polyhedra, and is not convenient if definitions of polygons in the clipped polyhedron are required. The polygon clipper approach

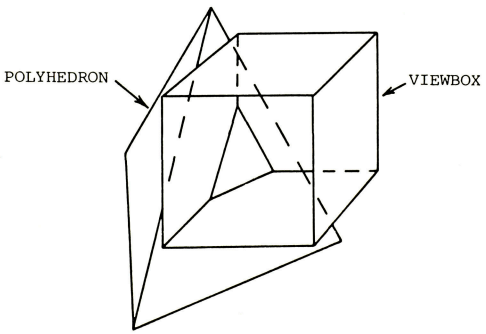


Figure 15 Polyhedron having no edge inside viewbox

benefits from a recursive implementation, would handle non-convex polyhedra, and would directly yield polygons in the clipped polyhedron. If either of these latter two facilities were needed then the polygon clipper should be used. However, the restricted requirements of the present application, together with avoidance of a recursive implementation (which is relevant when using FORTRAN), led to its choice in the implemented algorithm.

Subdivision

Much of the foregoing has assumed that objects, or rather their enclosing polyhedra, are disjoint. For many scenes the choice of objects can be made to ensure that this is the case. However, this is not always possible (e.g. an automobile entering a tunnel), and even where it is, the priority graph can contain cycles which thwart any attempts to construct a priority list. For completeness it would also be desirable to be able to handle intersecting objects. The proposed solution to all these problems is subdivision of the objects involved.

Subdivision can be done in many ways, depending on the object representation used by the model procedure. In the case of objects represented by groups of polygons, subdivision can be realized by bisecting the object into two pieces with a plane. The two objects so formed have the convenient property that they are separable by the plane which split them apart.

For objects which are simple collections of other objects subdivision is a logical process which rearranges the group of objects into two groups. The two groups so formed may not be separable by a plane.

Objects represented by bivariate parametric patches can be subdivided parametrically, any other type of subdivision being considerably more difficult. Again, the two fragments formed will not, in general, be separable by a plane.

In view of the many techniques for subdivision, it is necessary that subdivison be one of the functions carried out by the procedure models. In subdividing, an object may be transformed from one type into another. For example, a sphere cannot be subdivided into two spheres. In this case the sphere must either be replaced with two hemisphere objects, or else turn itself into polygons and be split by a polygon splitting algorithm, and from there after always be treated as polygon objects.

This latter method is the universal solution to the subdivision problem. If a procedure model is capable of representing its subject as a collection of polygons then it can always be subdivided. The requirement that an object can ultimately be represented by polygons is not necessarily an extra constraint in that it is expected that most procedure models will use a polygon based algorithm for the purpose of generating images of their subjects.

The following examples show how subdivision can solve the various

problems mentioned above, namely non-separable polyhedra and intersecting objects.

Figure 16a shows two non-intersecting objects whose enclosing polyhedra intersect. If the arch is subdivided as shown in Figure 16b, the three resulting fragments are separable.

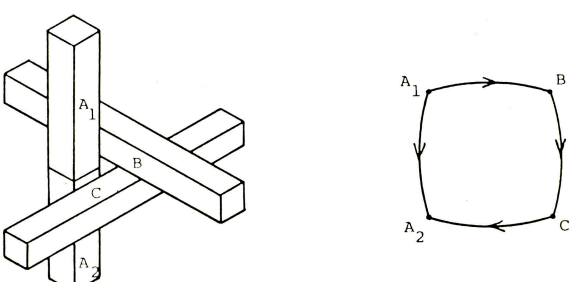
An example of non-orderable polyhedra, due to a cycle in the priority graph, was illustrated in Figure 9. A suitable subdivision of any of the three objects can break the cycle. Such a subdivision is shown in Figure 17, which generates the priority order A1,B,C,A2.

The problem of intersecting objects, which may be of different types, is treated next. The enclosing polyhedra of two intersecting objects will, of necessity, intersect, a fact that will be discovered when an attempt is made to find a plane which separates them. Consequently one, or both, of the objects will be subdivided, and the fragments treated as individual objects. However, since the objects themselves intersect, this process will repeat indefinitely. Consequently, the following action is proposed.

When two conflicting object fragments have been subdivided a sufficient number of times, they will both be transformed into polygon objects and treated as a single object. The intersection will then be treated by the polygon object model procedure when it generates an image of the compound object. The "sufficient number of times" will be determined either by fragment size, the size of the image of the

a b

Figure 16 Resolution of intersecting polyhedra



a b

Figure 17 Removal of cycle in priority graph

fragment, or by the number of times the original object has been subdivided.

This approach is in keeping with the philosophy of the algorithm in that the mainstream of the algorithm seeks to solve the visible surface problem at a macro level, and is not concerned with the details of any one object. The mechanism puts the task of handling the intersection onto the model procedures, where, it is felt, it belongs.

Frame Buffer

The frame buffer used in the implementation of the priority algorithm assembles the image in terms of individual picture elements. Two versions were used, one giving 512 x 512 picture elements and the other giving 1024 x 1024 picture elements. Even with only 8 bits per picture element to store intensity the amount of memory this represents is larger than can be accommodated in the main memory of the computer being used, a 64K PDP-10. Consequently, the frame buffer was implemented on disk and is paged into main memory for use.

The pages used are thin horizontal stripes the full width of the picture. By using an intelligent, but simple, paging algorithm the disk latency time is reduced to acceptable limits. It was expected that procedure models would generate their images using a scan line algorithm, so the paging strategy simply ensures that the page next to the one being used is set up in memory, using double buffers.

The choice of a picture element frame buffer is probably the lowest level attainable, and all procedure models should be able to generate images in the form of individual picture elements. A segment frame buffer was implemented to handle visible segments such as those generated by Watkins' algorithm. Such a frame buffer was also used by Newell et al (8). It was found that although the segment buffer was less demanding on memory space it was comparatively slow, and did impose an upper limit on image complexity. Also it could not handle images such as are generated by Catmull's (15) patch rendering algorithm.

The existence of a picture element frame buffer allows several rather nice features to be added to the system. The intensity value of zero is used to indicate that a picture element has never been written, care being taken never to generate this value as part of an image. Such a feature allows the generated image to be combined with various different backgrounds, ranging from a uniform intensity to a scanned-in photograph. This is done by putting the background into another frame buffer and then overwriting it with the generated image, taking care not to overwrite with zero. This facility can also be used to save much time in an animated sequence of images. If the background is static but the foreground is moving then a sequence of images can be generated by saving the background, whether computer generated or not, and then for each frame taking a new copy of the background and overwriting it with the newly generated foreground image.

This technique can be extended to several layers provided that the priority ordering of whole layers can be established. One possibility for determining this priority, and the contents of each layer, is to examine the priority lists of several frames for occurences of common groups of objects known to be static relative to the picture. Such an exercise is perhaps not so unreasonable if one notes that frame-to-frame coherence can be achieved by using the priority list from one frame as the initial list for the subsequent frame. As the priority ordering for the new frame is checked, any unchanged groups of static objects can be detected and separated out as potential static layers for subsequent frames. The development and implementation of these ideas is considered outside the scope of this paper.

Implementation Notes

This section presents some of the techniques used in the implementation of the algorithm. The implementation was written mainly in FORTRAN, with some assembly language routines where necessary. This choice was made mainly for reasons of portability, at the cost of some convenience. It was thought that the lack of recursion, list processing facilities, and clumsy overlay facilities would be a major problem, though this has proved not to be the case, and the system is capable of demonstrating most of the features described.

The procedure models are implemented using the structure described in Chapter VI and Figure 6-1, namely model procedure, model instance and object instance. In the absence of a convenient overlay facility the model procedures are loaded permanently into the system. If a new type of procedure is needed the system has to be reloaded. This restriction could be removed given a simple, one level overlay facility since the only routines to be overlayed are the model procedures.

The model and object instances are implemented using a stack in which each instance is represented by a contiguous block of words in the stack. Each model instance has a logical pointer to its model procedure, and each object instance has an actual pointer to its model instance. The term "stack" is not strictly correct, although space is allocated in sequential order. Many intermediate computations are carried out in true stack form using free space beyond the last allocated block. When a block is deleted it is simply marked as such. The structure is such that a simple garbage collection procedure can compress out the deleted blocks when necessary.

Each model instance holds its enclosing convex polyhedron in the polygons, edges and vertices format illustrated in Figure 13. The transformed, clipped polyhedra are not held with the associated object instances, but are built on top of the stack when needed, and are referenced by the object instances. The format of the transformed, clipped polyhedra is a list of plane vectors, a list of vertex vectors, and a list of pointers to the vertex vectors to denote the

two dimensional enclosing polygon, which is derived directly from the transformed, clipped polyhedron, not from the object itself. When an object has been established as having the next highest priority, the transformed, clipped polyhedron and polygon are marked as deleted. to conserve space.

The method used by all procedure models to generate images of their subjects is Watkins' algorithm. This implies that each procedure model must be capable of representing its subject as a group of polygons, although the whole group need never all exist at the same time.

The hardware Watkins processor and display devices used are connected to a single-user PDP-10 computer and therefore actual image generation is carried out on that machine, using private disk packs for the frame buffer. However, the generation of the priority ordered list of objects does not use any special hardware and so is normally carried out on a time-shared PDP-10, although it can also run on the single user machine. This has meant segmenting the system into two phases, the first one to generate the priority ordered list, and the second to generate the image. This subdivision has proved fairly convenient, although it necessitates saving the priority list between phases.

CHAPTER IX

EXAMPLES OF USE OF THE PRIORITY ALGORITHM

This chapter is included both to illustrate the types of scenes that can be processed by the priority algorithm described, and to give some examples of ways in which objects can be represented as procedure models. Classes of representations which have been implemented as procedure models include: collections of polygons; axisymmetric objects where the profile is represented as a list of vertices; spheres; collections of Bezier bicubic patches; parameterized office buildings; groups of objects each one being represented by any of the previous models; and automobiles represented as a special group defining half the body and two wheels. The use of FORTRAN prohibits groups from being defined as collections of groups.

Figure 18 shows an automobile body represented as a collection of polygons. This representation was generated by taking measurements from an actual automobile. It would be difficult to generate a higher level representation, bivariate patches for example, since much information has been lost in the polygonal representation. However, the fact that the automobile body is symmetric about a vertical plane may be exploited, since only half of the body need be stored.

Figure 18 Automobile body

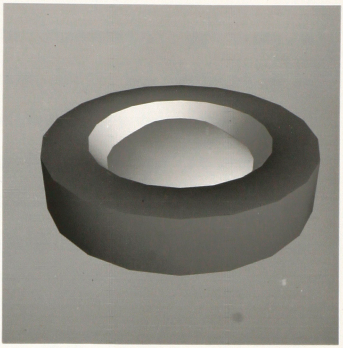


Figure 19 Wheel

Figure 20 Half body with wheels

Figure 21 Whole body with wheels

It was desired to put wheels on the automobile. By using a simplified representation, an automobile wheel was modeled as an axisymmetric object, Figure 19. The procedure model repesentation of axisymmetric objects has the capability for generating the number of sectors used to approximate the circular cross section, based on the size of the image of the object. This achieves economies in image generation of such objects.

In order to define an automobile with wheels, a group representation is used. The model includes a set of three references to the component parts of half an automobile, namely, half the body, the front wheel, and the rear wheel. Such a collection of components is shown in Figure 20. The generation of an enclosing convex polyhedron exploits the fact that automobiles are generally approximately box shaped, and do not normally get very close to other objects. Consequently, a minimum volume enclosing rectangular box is used, being generated from the maxima of the extrema of the three component parts, and accounting for both halves of the automobile.

Image generation of the whole automobile is carried out separately on the two halves. Knowing the equation of the symmetry plane, it is a simple matter to determine which half of the automobile is farthest from the eye. An image of this half is first generated and written into the frame buffer. Then an image of the nearer half is generated and written into the frame buffer, correctly overwriting any parts of the image of the first half. This process may be thought of as a special purpose priority algorithm tailored to this particular

class of objects. The whole automobile is shown in Figure 21.

The above two part image generation process could be applied to any object exhibiting mirror symmetry. However, even the simple representation of an automobile used here is not strictly symmetric, in that if the front wheels are turned then they should both turn in the same direction, Figure 22. Given the procedural representation of the automobile it is a relatively simple matter to include the angle of turn as a parameter, and to negate this parameter when generating the image of one of the halves. The inclusion of such a constraint in a more general purpose model of symmetric objects would be quite difficult.

As an example of a class of objects which can be described by a relatively small number of parameters, Figure 23 shows a row of buildings. Each building is described by 12 parameters to specify such things as number of floors, number of windows per floor on front and side, window dimensions, material properties, etc. The enclosing convex polyhedra are always rectangular boxes, and are generated by a simple arithmetic computation involving window dimensions, number of floors etc. Image generation exploits the fact that in any view at least two, possibly three, walls will not be visible. The determination of such invisible walls is determined by examining the plane equations of the individual walls. For any wall found to be invisible, no part of that wall is considered during the image generation process. This elimination of whole walls by a single

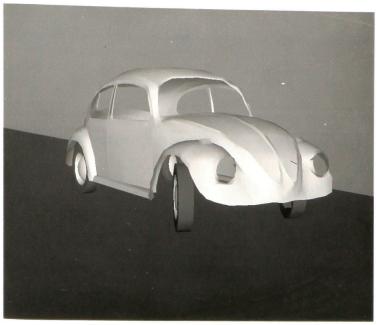


Figure 22 Wheels turned

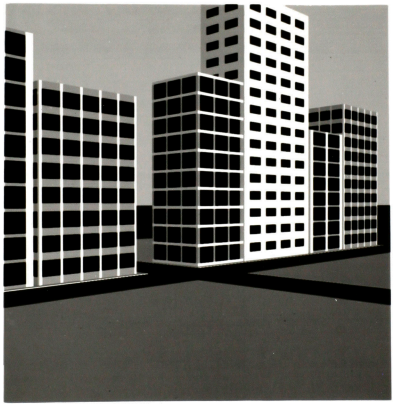


Figure 23 Row of buildings

operation makes possible considerable economies during image generation.

An important class of object representations which, it is felt, should be included in any three dimensional object processing system is the bivariate patch description of arbitrary curved surfaces. Such representations have been developed specifically to facilitate the description and modification of arbitrary shapes. It would seem desirable, therefore, to process such forms directly, as opposed to transforming them into some other representation. It transpires that the formulation of polynomial bivariate patches introduced by Bezier (20), and later generalized by Riesenfeld (14), lends itself directly to the requirements of the hidden surface algorithm presented.

In Bezier's formulation, a patch is specified in terms of a mesh of control points. An example of such a mesh is shown in Figure 24, for a bicubic patch. The mesh may be thought of as an approximation to the patch, the precise definition of the relationship being given by Bezier's formulation. Some lines in the patch corresponding to the mesh shown in Figure 24 are shown in Figure 25.

Details of Bezier's formulation are not given here. However, one important property of the relationship between the mesh and correponding patch will be given. This is that the patch is always entirely contained within the convex hull of the control points. Hence, if a convex polyhedron containing all the control points can be found, it can be used as the charaterizing polyhedron enclosing the

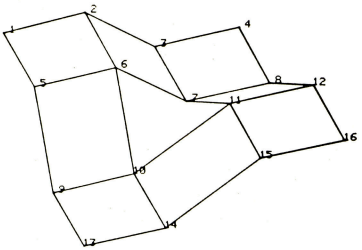


Figure 24 Mesh for Bezier patch

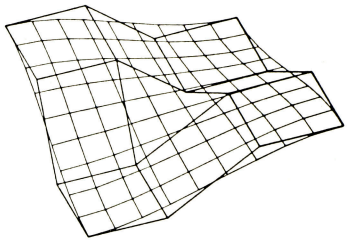


Figure 25 Mesh with patch

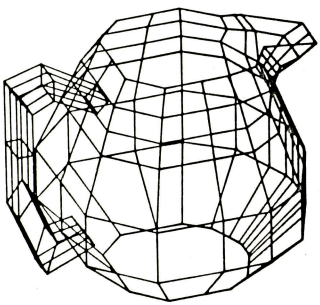


Figure 26 Meshes defining jug

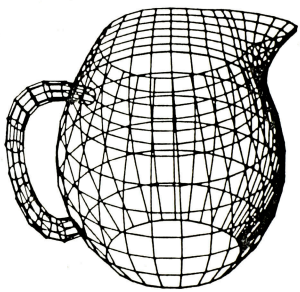
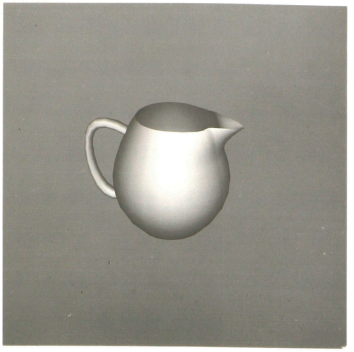


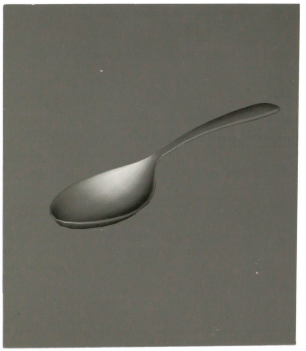
Figure 27 Parametric lines on jug

patch. Moreover, if the convex hull of the control points is used, its shape will resemble that of the convex hull of the patch itself. A simple extension of this result is that the convex hull of the control points of a collection of patches contains all the patches, and may therefore be used as the enclosing convex polyhedron for the collection of patches. This extended result solves an otherwise difficult problem, that of constructing an enclosing convex polyhedron which is a reasonable approximation to the convex hull of the curved shape. An example of a collection of meshes and the corresponding patches defining a small jug is shown in Figures 26 and 27.

As was mentioned earlier, the technique used to generate images in the currently implemented algorithm is to derive a polygonal approximation to the object then to use Watkins' algorithm. For bivariate patches the polygonal approximation is derived by splitting up each patch into a rectangular array in parametric space, then approximating each parametrically rectangular fragment with a quadrilateral constructed on the corners of the fragment. Examples of images generated using this technique are shown in Figure 28. As in the case of axisymmetrix objects, it is possible to vary the degree of approximation depending on the image size of each patch, although this has not been implemented. Another possibility for generating images of bivariate patches would be to use an algorithm developed specifically for the task. Catmull's (15) algorithm is an example.

Figure 29 shows an image of a scene involving multiple objects.





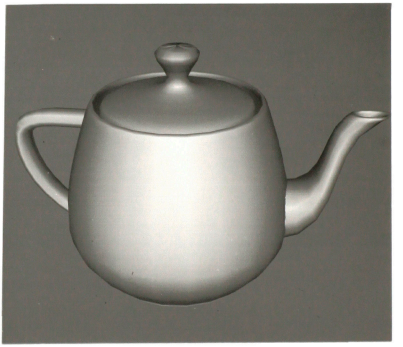


Figure 28 Various objects defined using Bezier patches

The various pieces of crockery, the spoons, and the drapes are modeled using bicubic Bezier patches. The teapot stand is modeled as an axisymmetric object, and the table top and mat are modeled using polygons. The bodies of the teapot and cups, and the saucers, could have been modeled as axisymmetric objects, but were not due to the absence of a design system capable of using these two forms together. A design system based on procedure models could, of course, have fulfilled this need.

Figure 30 shows an array of pawns on a large checkerboard. Each pawn is modeled as an axisymmetrix solid, and the checkerboard is modeled using polygons, though a more strongly procedural model may have been more appropriate for such a simply generated object. The fact that all the pawns are identical was not explicitly used by the algorithm, except that only one profile had to be stored. The number of polygons generated during the creation of this image is in excess of 180,000.

Finally, Figure 31 shows eight instances of an object modeled as a collection of polygons. Acknowledgements must go to Jim Clark who laboriously encoded this object by manual means. The carousel was modeled as three axisymmetric solids: the base, pole, and top.

Figure 29 Table setting

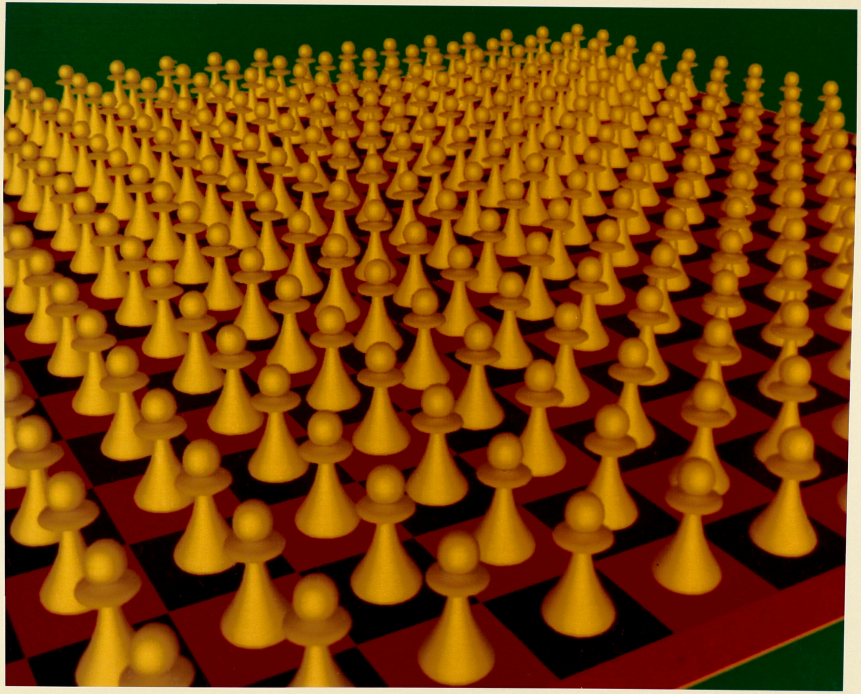


Figure 30 361 Pawns

Figure 31 Carousel

CHAPTER X

A CATEGORIZATION OF PROCEDURE MODELS

The types of procedure models found to be convenient in digital image synthesis form a highly varied class. At one extreme the procedure model can be rich in stored data, and the model procedure is essentially a data manipulator. At the other extreme the only stored data might be a few constants embedded in the model procedure, and the object, its image and properties, are all generated when required.

Another attribute which affects the classification of procedure models is the degree of parameterization used, and indeed, the range of objects which can be represented by the model procedure. The technique used to generate an image of the object is another factor by which procedure models may be distinguished.

In an attempt to define these various attributes the following list is given, together with a brief explanation and some examples.

Strongly Procedural - the model procedure generates properties and images of the object by computation rather than by access to a voluminous data base. Examples include regular geometric shapes such as spheres, cylinders etc. Objects defined using surface patches can be included in this category.

Data Rich - in some senses this is the opposite of Strongly Procedural though the use of non-trivial processing by the model procedure is not prohibited. This category is intended for

procedure models which represent their subjects by a fairly conventional numerical data base. An example of this type is the use of polygons to represent objects.

Generator - The model procedure does not represent any one object but provides the processing capability to generate an object from suitable parameters. A B-spline patch interpreter falls into this category in that given the control points for a collection of patches the model procedure can generate properties and images independent of what object the patches represent.

Strongly Parameterized - The form of the represented object can vary widely depending on the parameters, but the class of objects is known. An example in this category is a building generator which might have as parameters the type of building (office block, motel, home) and the number of floors.

In conjunction with the visible surface priority algorithm presented, two further categories of procedure models may be identified, based on the technique used to generate images.

Priority Image Generation - The technique used to generate images is a priority technique which can therefore use the frame buffer as an active element in generating the image. Image generators which use the algorithms of Schumaker et al, or Newell et al, come into this category.

Independent Image Generation - Effectively the opposite of Priority Image Generation. This category covers image generators which produce their images independently of the frame buffer.

Watkins' algorithm provides an example.

It is clear that any one procedure model might fall into several of these categories. However, such a categorization can be useful in describing a given or proposed procedure model, and can be used to generate guidelines as to just how a required procedure model should be structured.

CHAPTER XI

CONCLUSIONS

The properties of procedure models as applied to the representation of three dimensional objects, for the purpose of synthesizing images in the form of shaded pictures, have been investigated. It has been shown that procedure models facilitate the processing of scenes of far greater complexity than has proved practicable using data base modeling techniques. The generality and flexibility of procedure models has enabled a system to be implemented which can be, and has been, incrementally expanded to accomodate new model formulations.

It is believed that the benefits of procedure models are not confined to the field of image synthesis, but have considerable relevance in many areas where modeling of three dimensional objects is of concern, such as computer aided design, computer aided manufacture, stress analysis, dynamics simulation, etc. The investigation of this hypothesis, while outside the scope of this paper, should provide a stimulating, and hopefully fruitful, research project.

APPENDIX A

ENCLOSING CONVEX POLYGON

Given a set of points, P, in two dimensions the problem is to find the minimum area convex polygon enclosing all the points, i.e. the convex hull. An algorithm, suggested by Rudolph Krutar, is given.

The points, P, are considered one at a time in a serial manner. Suppose, at some stage of the algorithm, that the convex hull of the first i-1 points in P has been found. Let this polygon be represented in terms of its vertices and the line equations of its edges, defined such that all line normals point inside the polygon. Point Pi is next considered.

Pi is checked against each line equation to determine on which side of each line it lies. If Pi lies on the inside of every line then it is inside the current polygon and is not considered further. However, if Pi lies outside at least one line, then the current polygon must be extended to embrace Pi.

Consider the situation shown in Figure 32. Pi is found to lie outside the lines corresponding to edges E2, E3, and E4. Due to the convexity of the polygon the set of edges for which Pi is outside will always be consecutive edges of the polygon, in a cyclic sense. The extension of the polygon to embrace Pi involves replacing the set of

edges having Pi outside with two new edges, formed from Pi and the polygon vertices at the ends of the set. In Figure 32 this involves replacing edges E2, E3 and E4 with edges V2-Pi and Pi-V5.

The above process is repeated until all vertices of P have been considered, at which time the current polygon is the convex hull of the set P.

To start the algorithm an initial two sided polygon, constructed on any two non-coincident points, is created. A better starting polygon can be achieved by using the two points having maximum separation in one of the coordinate directions.

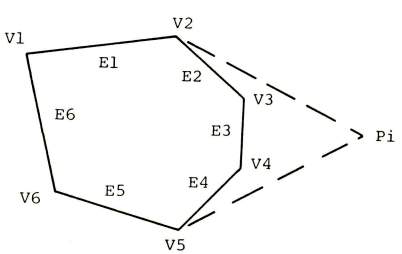


Figure 32 Current polygon and test point Pi

APPENDIX B

ENCLOSING CONVEX POLYHEDRON

Given a set of points, P, in three dimensions, the problem is to find the minimum volume convex polyhedron enclosing all the points, i.e. the convex hull.

The method used is a direct extension of that given for enclosing convex polygons in Appendix A. The current polyhedron is represented by its vertices and plane equations of its faces. Each test point, Pi, is checked against each plane equation to determine the set of faces for which Pi is outside. The replacement of this set of faces, in order to embrace Pi, is not so straightforward as in the two dimensional case, since there is no simple ordering of the faces. It is necessary to pair adjoining edges of faces of the set to find the boundary edges, which are the non-paired edges. The new faces are then generated using Pi and each edge of the boundary in turn.

To start the algorithm an initial two-sided polyhedron constructed on any three non-colinear points is created.

APPENDIX C

TWO DIMENSIONAL SEPARATOR THEOREM

The theorem states that if two convex polygons in the plane have no area in common then at least one edge of at least one of the polygons is a segment of a line which separates the two polygons.

In spite of the apparent obviousness of the correctness of this theorem, no correspondingly simply proof has yet come to the attention of the author. Proof is essentially by construction.

Consider a line joining two points, one inside each of the given polygons, A and B. The centroids of the vertices of each polygon provide one such pair of points. Now consider moving the two polygons towards each other along this line. The two polygons will touch in one of three types of configurations, illustrated in Figure 33. A key edge will be defined for each these cases.

Consider cases a and b. In case a the key edge is the edge of A making contact with a vertex of B. In case B, the key edge is either of the edges in contact. It is proposed that the key edge, as defined above for cases a and b, is the sought after edge whose extension separates the two polygons in their original positions. Clearly, the key edge has the effect of separating the two polygons in their touching positions. If the two polygons are moved back to their original positions, the polygon not containing the key edge must move away from the key edge, thereby maintaining the separation of the two

polygons by the key edge extended to a line.

Case c is considered separately. Four edges meet at the point of contact. Consider either pair of opposite edges, e.g. E1,E3 or E2,E4 in Figure 33. One polygon is inside the angle formed by the two edges, and the other polygon is outside. The key edge is the edge belonging to the polygon outside the angle. If the edges are colinear then either edge may be used as the key edge. The argument that the key edge is the sought after edge whose extension separates the two polygons is the same as for cases a and b.

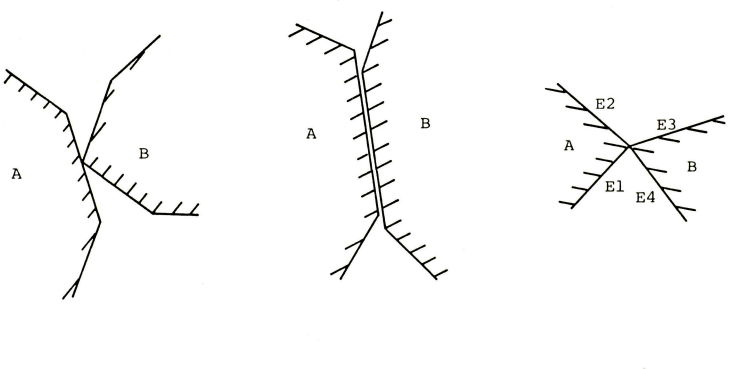


Figure 33 Contact between two polygons

LIST OF REFERENCES

1. Sutherland, I.E.; Sproull, R.F.; and Schumaker, R.A. "A Characterization of Ten Hidden-Surface Algorithms." ACM Computing Surveys 6 (March 1974).

2. Sutherland, I.E., and Hodgeman, G.W. "Reentrant Polygon Clipping." Comm. ACM 17 (January 1974):32.

3. Mahl, R. "Visible Surface Algorithms for Quadric Patches." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report UTEC-CSc-70-111, December 1970.

4. MAGI, Mathematical Applications Group Inc. "3-D Simulated Graphics." Datamation 14 (February 1968):69.

5. Warnock, J.E. "A Hidden-Line Algorithm for Halftone Picture Representation." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report TR 4-15, 1969.

6. Watkins, G.S. "A Real-time Visible Surface Algorithm." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report UTEC-CSc-70-101, June 1970.

7. Schumaker, R.A.; Brand, B.; Gilliland, M.; and Sharp, W. "Study

for Applying Computer-Generated Images to Visual Simulation." US Airforce Human Resources Laboratory, Tech. Report AFHRL-TR-69-14, September 1969.

8. Newell, M.E.; Newell, R.G.; and Sancha, T.L. "A Solution to the Hidden Surface Problem." Proc. ACM National Conf. (August 1972): 443.

9. Romney, G.W.; Wylie, C.; Evans, D.C.; and Erdahl, A. "Halftone Perspective Drawings by Computer." Proc. AFIPS FJCC 31 (1967).

10. Bouknight, W.J. "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Representations." Comm. ACM 13 (September 1970).

11. Gouraud, H. "Computer Display of Curved Surfaces." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report UTEC-CSc-71-113, June 1971.

12. Bui-Tuong, P. "Illumination for Computer-Generated Images." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report UTEC-CSc-73-129, July 1973.

13. Roberts, L.G. "Machine Perception of Three-Dimensional Solids." MIT Lincoln Laboratory, Tech. Report TR 315, May 1963.

14. Riesenfeld, R.F. "Applications of B-spline Approximation to Geometric Problems of Computer-Aided Design." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report UTEC-CSc-73-126, March 1973.

15. Catmull, E. "A Subdivision Algorithm for Computer Display of Curved Surfaces." Dept. Comp. Sci., Univ. of Utah, Salt Lake City, Tech. Report UTEC-CSc-74-133, December 1974.

16. Hewitt, C.; Bishop, P.; and Steiger, R. "A Universal Modular ACTOR Formalism for Artificial Intelligence." Proc. Third Int. Joint Conf. on Artificial Intelligence (August 1973):235.

17. Winograd, T. Understanding Natural Language. New York: Academic Press, 1973.

18. Birtwistle, G.M.; Dahl, O.J.; Myhrhaug, B.; and Nygaard K. SIMULA BEGIN. Philadelphia, AUERBACH Publishers Inc., 1973.

19. Smith, J.M., and Chang, P.Y. "Optimizing the Performance of a Relational Data Base Interface." Paper presented at SIGMOD Workshop, San Francisco, May 1975.

20. Bezier, P. Numerical Control-Mathematics and Applications. Translated by R. Forrest. London: John Wiley and Sons, 1972.